TIUP: Effective Processor Verification with Tautology-Induced Universal Properties

Yufeng Li[1,2], Yiwei Ci[1,2], Qiusong Yang[1,2]
[1]Institute of Software, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
qiusong@iscas.ac.cn

Background: formal verification is necessary for processor design

- **Pentium FDIV Bug (1994)**
  - **FDIV: floating point division unit**
  - **Certain floating point division operations performed produced incorrect results**
  - **Byte magazine estimated a 1 in 9 billion chance of inaccurate results from floating point divides with random parameters**
  - **Intel spent $475 million replacing the flawed processors, causing significant damage to its reputation.**

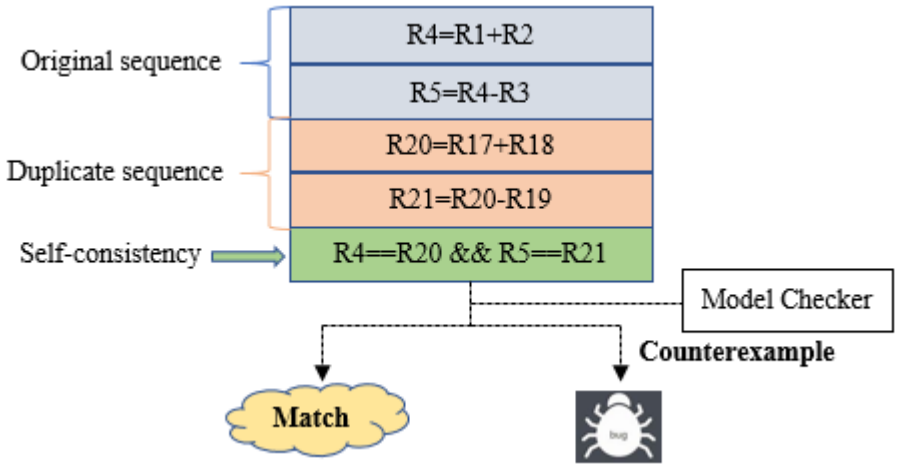
Background: specific property description is inefficient

- **Problem**
  - **Design-dependent**
  - **Processor operations and control logic are complex, requiring intricate properties written using advanced SVA features.**
  - **The interaction between combinational and sequential logic requires consideration**
  - **Writing properties is time-consuming and error-prone**

```
property check_sub_instruction;

assume:
    at t: branch_flag = '0';
    at t: instruction = SUB;
    at t: pipeline_stage = decode;

prove:
     at t+1: pipeline_stage = execute;
     at t+1: result = op1 - op2;
     at t+2: regfile_wb = result @ t+1;

endproperty
```

Background: universal property

- **Simplified properties formulation**
  - **Design-independent**
  - **Greatly reduced the threshold of formal verification**
- **Self-consistency property**



  - **The execution of an original instruction should yield consistent results with the execution of its duplicate instruction**
  - **Industrial cases: automotive microcontroller cores, stand-alone hardware accelerators for AI**

Motivation

- **A single universal property is not sufficient for verification**
  - **False positives in verifying single-instruction bugs**

```
// Original instruction sequence
R4 = R1 + R2      ⇒    R4 = R1 − R2
R5 = R4 + R3      ⇒    R5 = R4 − R3

// Duplicate instruction sequence
R20 = R17 + R18   ⇒    R20 = R17 − R18
R21 = R20 + R19   ⇒    R21 = R20 − R19

BNE  R4, R20  ERROR_DETECTED
```

Original sequence	*Duplicate sequence*
$R0 = 0,\quad R4 = 1,\quad R5 = 2$	$R16 = 0,\quad R20 = 1,\quad R21 = 2$
$0 : R3 = R5 - R4$	$0 : R19 = R21 - R20$
$1 : BEQ\quad R3, R0\quad \#5$	$1 : BEQ\quad R19, R16\quad \#5$
$//\,R3\quad != 0$	$//\,R19\quad != 0$
$//\,Branch\ \ not\ \ taken$	$//\,\mathbf{Error : Branch\ \ taken}$
$2 : \mathbf{R2 = R4 + R5}$	$//Ignore\ \ QED\ \ instruction$
	$(2 : R18 = R20 + R21)$
$//\,\mathbf{R2 = 3}$	
	$//Use\ \ \mathbf{Non\text{-}QED}\ \ instruction$
	$\mathbf{5 : lui\ \ R18,\ \ 0}$
	$//\,\mathbf{R18 = 0}$

  - **The single self-consistency property is too generic, causing state explosion issues easily and shallow BMC unfolding depth.**

- **Idea: expanding a set of universal properties**

Tautology-induced universal properties

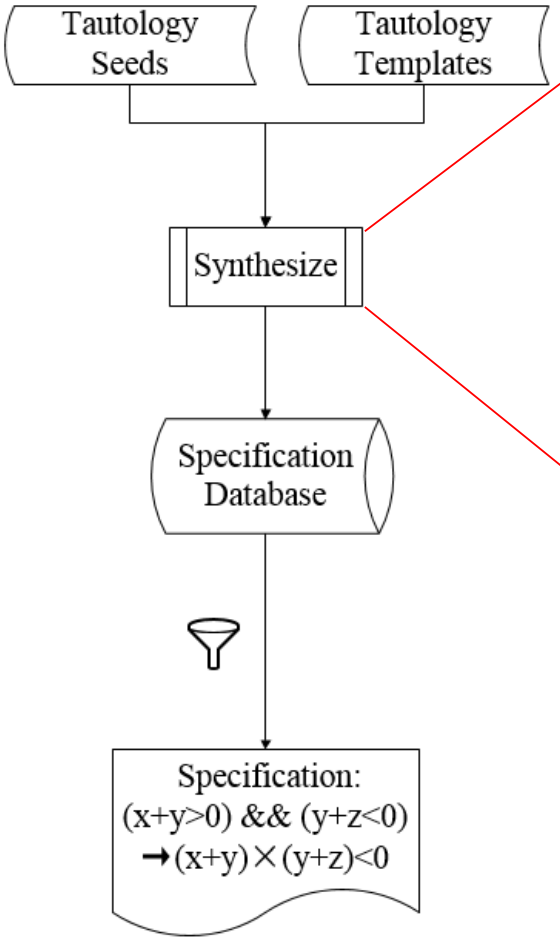
$CPU \vDash Tautology$

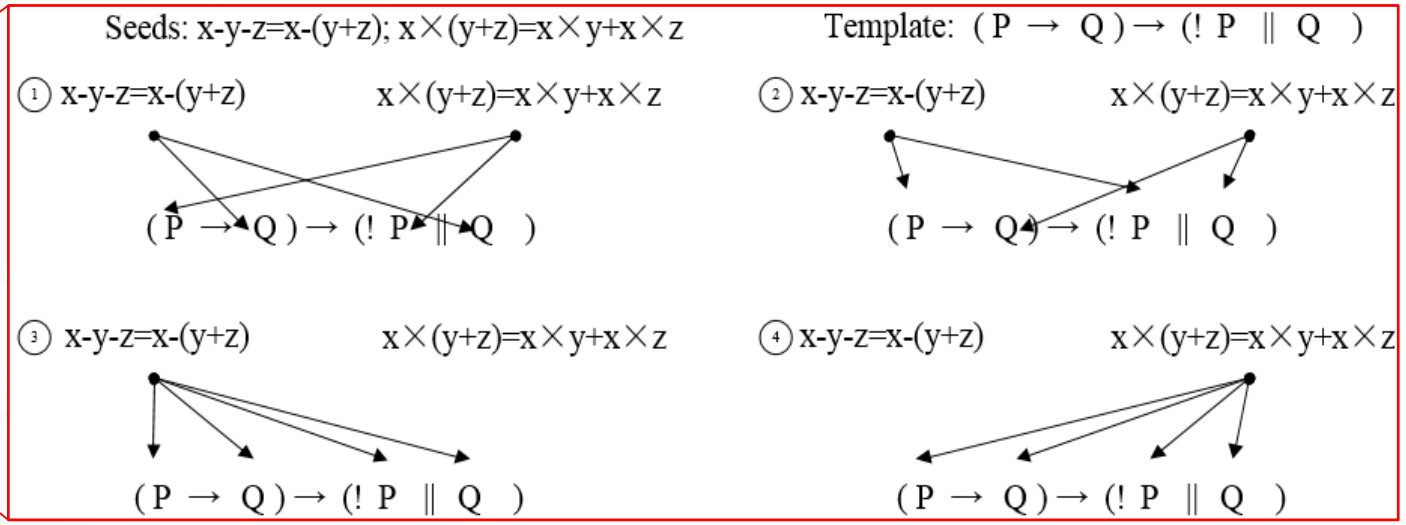
- CPU = CU + ALU
  - CU: control unit
  - ALU: arithmetic logic unit

- Self-consistency ⇔ Identity law ($A \equiv A$)
  - If two instances have the same initial state and transition paths, then the resulting state should also be identical

- Tautology: a statement that is always true
  - Commutative law: $A + B \equiv B + A, A \times B \equiv B \times A$
  - Associative law: $(A \times B) \times C \equiv A \times (B \times C)$
  - De Morgan law: $!(A\&\&B) \equiv (!A)||(!B), !(A||B) \equiv (!A)\&\&(!B)$
  - Distributive law: $(A + B)\%C \equiv (A\%C + B\%C)\%C$
  - Implication: $(A + B > 0)\&\&(B + C < 0) \rightarrow (A + B) \times (B + C) < 0$

Construct the set of universal properties

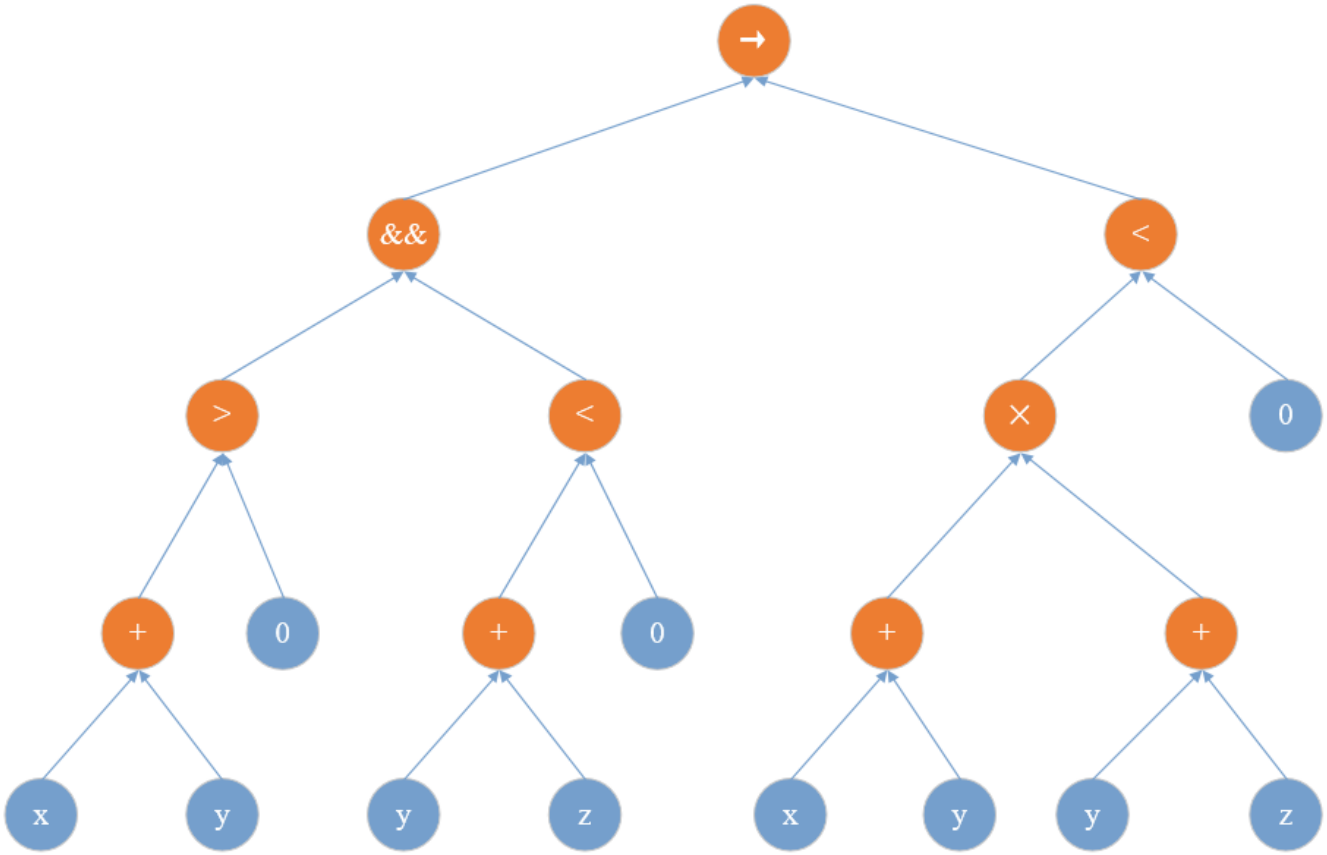




- **Synthesize**
  - **Seeds are first-order tautologies that encompass universal properties pertaining to processor's basic functions**
  - **Templates are propositional tautologies, e.g. , $(P \rightarrow Q) \rightarrow (!P||Q)$, P, Q is true or false**
  - **Non-logical connective of a template can be replaced by seeds**

Abstract syntax tree of universal properties

$$\vDash (x+y)>0 \&\& (y+z)<0 \rightarrow (x+y)\times(y+z)<0$$



- **Structure**
  - **Leaf node: constants and variables**
  - **Intermediate node: predicates, functions and connectives**

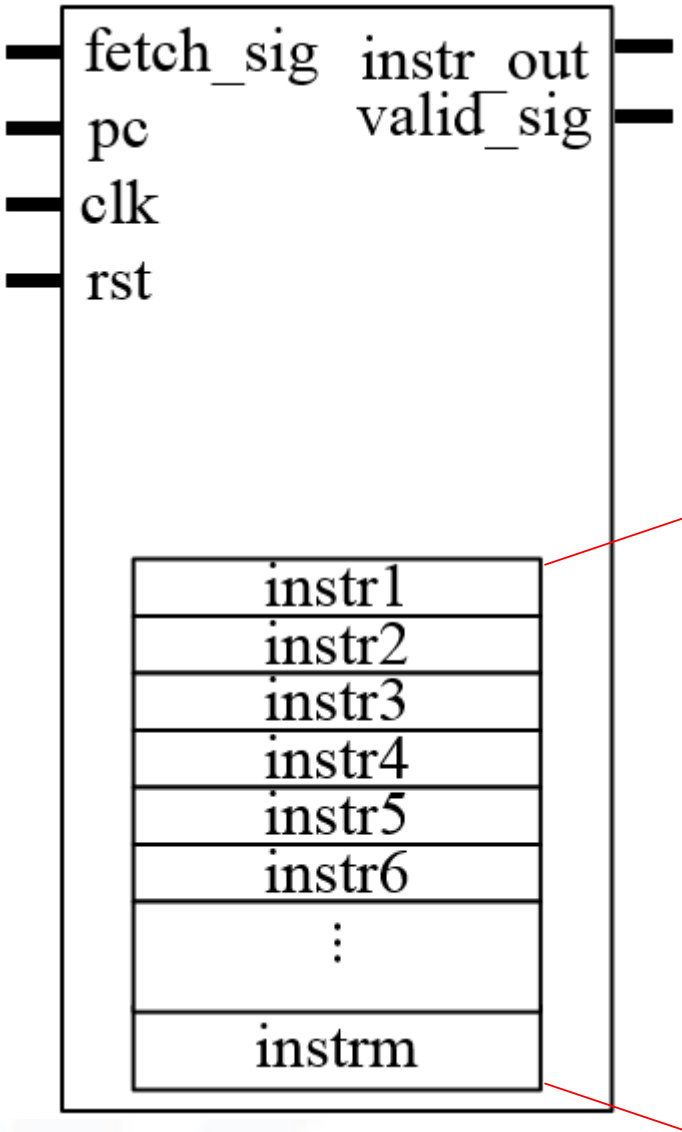
Intermediate representation

$$\vDash (x+y)>0 \&\& (y+z)<0 \rightarrow (x+y)\times(y+z)<0$$

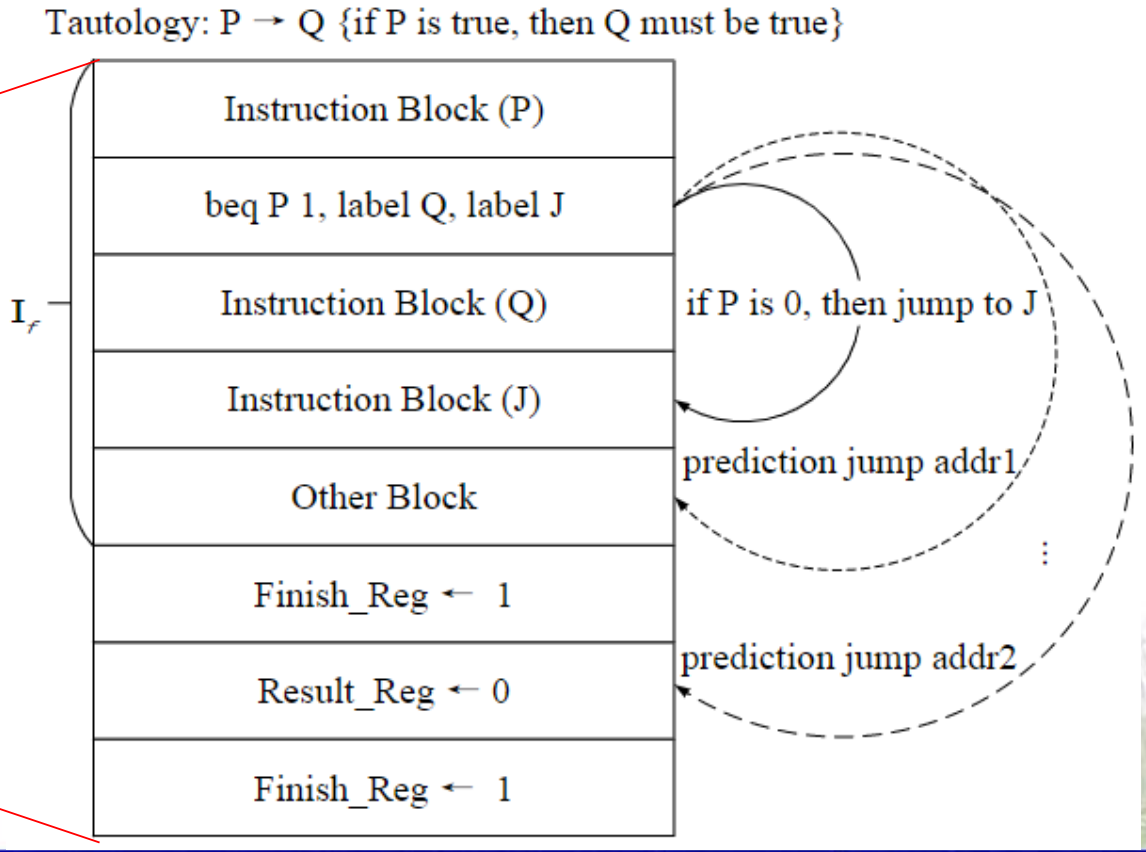
%tmp1 = add i32 %x, %y	+
%tmp2 = sgt i32 %tmp1, 0	>
%tmp3 = add i32 %y, %z	+
%tmp4 = slt i32 %tmp3, 0	<
%tmp5 = logic_and i32 %tmp2, %tmp4	&&
beq %tmp5 1, label %if, label %else	→
if:	
%tmp6 = mul i32 %tmp1, %tmp3	×
%tmp7 = slt i32 %tmp6, 0	<
%result_reg = and i32 %tmp7, %result_reg	&
else:	
other instruction blocks	

- **Control flow**
  - **If the code in the condition section evaluates to 1 (true), the control should jump to the code in the consequent section (if), and the result of executing the code in the consequent section should also be 1 (true).**
  - **$\%result\_reg$: the result flag register is initialized to 1**

Scheduler



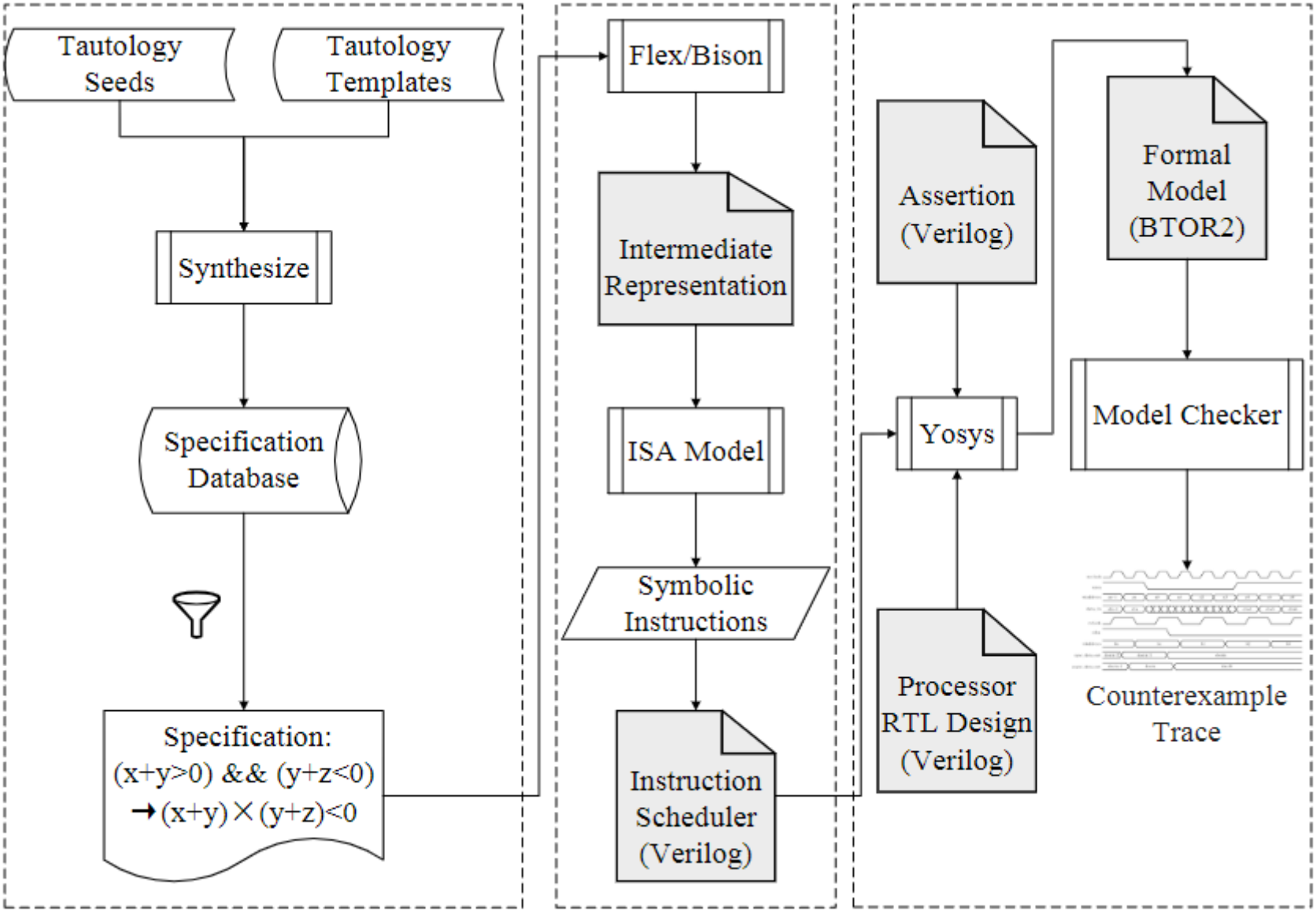
- **Port**
  - **fetch_sig: fetch signal**
  - **pc: program counter**
  - **instr_out: instruction output**
  - **valid_sig: output valid signal**



Assertion

- $Finish\_Reg \rightarrow Result\_Reg$
  - No expertise needed
  - No effort-intensive

Overview


Tautology Seeds
Tautology Templates
Synthesize
Specification Database
Specification:
(x+y>0) && (y+z<0)
→ (x+y) × (y+z)<0
Flex/Bison
Intermediate Representation
ISA Model
Symbolic Instructions
Instruction Scheduler (Verilog)
Assertion (Verilog)
Formal Model (BTOR2)
Yosys
Model Checker
Processor RTL Design (Verilog)
Counterexample Trace

Evaluation

No.	Synopsis	Category	Processor		Universal property-based method		
			RIDECORE	BIRISCV	$SQED$	S^2QED	$TIUP$
$a01$	No implementation of divide-by-zero exception (Return $0xffffffff$)	single	×	✓	×	×	✓
$a02$	No implementation of the division-modulo execution unit	single	✓	×	×	×	✓
$a03$	Register target redirection	multiple	✓	✓	✓	✓	✓
$a04$	Register source redirection	multiple	✓	✓	✓	✓	✓
$a05$	Incorrect unsigned operand less-than compare	single	✓	✓	×	×	✓
$a06$	GPR0 can be assigned	multiple	✓	✓	✓	✓	✓
$a07$	Incorrect instruction fetched after dispatch stall	multiple	✓	✓	✓	✓	✓
$a08$	Incorrect instruction fetched after an LSU stall	multiple	✓	✓	✓	✓	✓
$a09$	One of the buggy RS-m entries corrupted the MULH/MULHU instruction	multiple	✓	×	✓	✓	✓
$a10$	Erroneous branch addresses	single	✓	✓	×	×	✓
$a11$	Erroneous branch directions	single	✓	✓	×	×	✓
$a12$	Error in decoding next instruction's operand	single	✓	✓	×	×	✓
$a13$	Processor incorrectly decodes the next instruction to a NOP instruction	multiple	✓	✓	✓	✓	✓
$a14$	The value of the next register read is corrupted to all 0's	multiple	✓	✓	✓	✓	✓
$a15$	Erroneous speculative instruction aren't flushed	single	✓	✓	×	×	✓
$a16$	Unsigned multiply operand converts to signed	single	✓	✓	×	×	✓
$a17$	Source operand is misidentified as 0	multiple	✓	✓	✓	✓	×
$a18$	ALU opcode does not match with the actual circuit	single	✓	✓	×	×	✓
$a19$	Error in determining whether instructions in decode queue have been popped	multiple	×	✓	✓	✓	✓
$a20$	Logical error in fetch instruction valid signal	multiple	✓	✓	✓	✓	✓
Detect single-instruction anomalies					×	×	✓
Detect multiple-instruction anomalies					✓	✓	✓
Runtime (with anomalies) [min, max]					$[< 60s, > 1.5h]$	$[< 60s, > 1.5h]$	$[< 90s, < 988s]$
Runtime (without anomalies)					Timeout	Timeout	$< 988s$
Counterexample length ([min, max] instructions)					$[2, 14]$	$[2, 7]$	$[3, > 14]$

- **Observation**
  - **Capable of detecting all types of logic bugs**
  - **Reduced the state space and accelerated the solving speed**

Future Work

- **Redundancy of coverage for universal properties**
  - **Discovery of coverage gaps through mutation testing and exploration of a minimized set of universal properties capable of achieving complete coverage**

- **Universal properties regarding system control and I/O access**

Conclusion

- **Proposed a processor formal verification method based on universal properties**
  - **Construction method for a set of design-independent universal properties**
- **Implemented an automated framework for formal verification based on universal properties**
  - **Eliminated the tedious process of property formulation, reduced the barrier to entry for formal verification, and improved its efficiency**

Questions & Answers

Thanks !

Presenter: Yufeng Li